First Steps Towards Cumulative Inductive Types in CIC: Extended Version

Amin Timany *Bart Jacobs*

Report CW 684, March 2015

KU Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

First Steps Towards Cumulative Inductive Types in CIC: Extended Version

Amin Timany *Bart Jacobs*

Report CW 684, March 2015

Department of Computer Science, KU Leuven

Abstract

We discuss our on-going research on making inductive types cumulative in the predicative calculus of inductive constructions (pCIC) – the logic of the Coq proof assistant. Having inductive types be cumulative alleviates some problems that occur while working with large inductive types, e.g., the category of small categories, in pCIC.

We present the pCuIC system which adds cumulativity for inductive types to pCIC and briefly discuss some of its properties and possible extensions. We, in addition, give a justification for the introduced cumulativity relation for inductive types.

First Steps Towards Cumulative Inductive Types in CIC: Extended Version

Amin Timany and Bart Jacobs

iMinds-DistriNet, KU Leuven
firstname.lastname@cs.kuleuven.be

Abstract. We discuss our on-going research on making inductive types cumulative in the predicative calculus of inductive constructions (pCIC) – the logic of the Coq proof assistant. Having inductive types be cumulative alleviates some problems that occur while working with large inductive types, e.g., the category of small categories, in pCIC.
We present the pCuIC system which adds cumulativity for inductive types to pCIC and briefly discuss some of its properties and possible extensions. We, in addition, give a justification for the introduced cumulativity relation for inductive types.

1 Introduction

The predicative calculus of inductive constructions (pCIC), the underlying logic of the proof assistant Coq, has recently been extended to support universe polymorphism [5]. They extend the calculus of constructions with support for universe polymorphic definitions. They treat inductive types by considering copies of them at different universe levels – so long as levels satisfy constraints imposed by the inductive type and the environment. In this system the simple definition for a category,

```
Class Category: Type := {O: Type; H: O →O→Type;...}
```

defines a type $\texttt{Category}_{ij}$ where i is the universe level for objects and j is the universe level for homomorphisms. This allows a straightforward definition of the category of small[1] categories,

```
Instance Cat: Category := {O:= Category; H:= Functor;...}
```

which defines a term of type category, $\texttt{Cat}_{ijkl}$:$\texttt{Category}_{ij}$, with object type: $\texttt{Category}_{kl}$[2]. However, inductive types, e.g., `Category` not being cumulative, means that having a term `t`: $\texttt{Category}_{kl}$ such that `t` : $\texttt{Category}_{k'l'}$— is possible if and only if $k = k'$ and $l = l'$.

This side condition, however, has undesirable consequences. First and foremost, the term `Cat` above is not the category of all small categories, rather all categories at *some* lower universe level. On the other hand, statement of facts about `Cat` imposes restrictions on its universe levels. That is, only those copies of `Cat` are subject to the stated fact that conform to the restrictions imposed.

For instance, showing that the trivial category (a category with a single object and its identity arrow) with object type `unit`: $\texttt{Type}_0$ is the terminal object of $\texttt{Cat}_{ijkl}$, implies $k = 0$. Also, showing that $\texttt{Cat}_{ijkl}$ has exponentials (functor categories) implies $j = k = l$. The latter restriction is inconsistent with the restriction $n < m$ on $\texttt{TypeCat}_{mn}$, the category of types and functions in Coq. This means, a copy of `Cat` can't both have exponentials and a copy of `TypeCat` in its objects. Furthermore, having $\texttt{Cat}_{ijkl}$ cartesian closed restricts it so that $j = k = l = 0$.

[1] Here, smallness and largeness are to be understood as relative to universe levels.
[2] Subject to side constraints on universe levels, e.g., $k, l < i$.

It is, furthermore, noteworthy that such issues are not particular to category theory and are rather prevalent in any case incorporating large inductive types. Take the well-known definition of sets in type theory with inductive types:

`Inductive Ens: Type :=ens : Π(A: Type),(A → Ens) → Ens.`

In this case, $\texttt{Ens}_i$: $\texttt{Type}_{i+1}$ has constructor $\texttt{ens}_i$: Π (A: $\texttt{Type}_i$),(A → $\texttt{Ens}_i$) → $\texttt{Ens}_i$. As a result, the ensemble of small ensembles, ens Ens (λ(x: Ens). x), can't be formed as x in the body of the lambda-term is at a strictly lower universe level than the result ensemble.

To solve these problems, explicit lifting functions, e.g., $\texttt{Lift\_Ens}$: $\texttt{Ens}_i \to \texttt{Ens}_j$ with $i < j$, could be used. They allow formation of terms such as the ensemble of *lifted* small ensembles. However, such liftings involve case analysis on their input and hence working with them is vexing in practice. For, not only can't we prove, or even specify, Π(t: T), t = Lift_T t, but also proving the next best specifiable statement, Π(t: T), Lift_T (Lift_T t) = Lift_T t, would require case analysis on t. As a result, such equalities are not definitional and are thus not available to the core type checker, making working with types depending on such lifted values in particular very complicated.

The rest of this paper is structured as follows: in Section 2 we present pCuIC, an extension of pCIC with cumulative inductive types and discuss that it does not suffer from the problems just mentioned regarding category of small categories and ensemble of small ensembles. In Section 3, we discuss properties of pCuIC. Section 4 is devoted to lpCuIC, a subsystem of pCuIC, and its properties. There we will discuss that lpCuIC, like pCuIC, does not suffer from the aforementioned problems. In Section 5, we conclude with discussing possible extensions to the presented system.

2 pCIC with Cumulative Inductive Types (pCuIC)

In this section we present pCuIC, an extension of the predicative calculus of inductive constructions (pCIC) which additionally supports cumulativity for inductive types. The definition of pCuIC is identical to that of pCIC, except for cumulativity rule C-Ind. The following grammar depicts the form of terms in pCuIC.

$$
\begin{aligned}
& x, y, z, \ldots X, Y, Z, \cdots \in \mathit{Vars} \\
& i \in \mathbb{N} \\
t \ ::= \ & x \mid \texttt{Type}_i \mid \texttt{Prop} \mid \Pi x : t.t \mid \lambda x : t.t \mid t\ t \mid \mathsf{Ind}(X : t)\{t, \ldots, t\} \mid \\
& \mathsf{Constr}(i, t) \mid \mathsf{Elim}(t, t)\{t, \ldots, t\} \\
\Gamma \ ::= \ & \cdot \mid \Gamma, x : t
\end{aligned}
$$

In this grammar, Var is the set of variables, t denotes terms of the language and Γ denotes typing contexts. In the sequel, we use $t, m, \ldots, A, \ldots, M, N, T, \ldots$ for terms, $i, j, \ldots$ and for natural numbers and s to stand for a sort, i.e., $\texttt{Type}_i$ or $\texttt{Prop}$.

In this language variable x in terms $\Pi x : A.B$, $\lambda x : A.t$ and variable X in $\mathsf{Ind}(X : A)\{C_1, \ldots, C_n\}$ are bound respectively in B, t and $C_1, \ldots, C_n$. A variable that appears in a term but is not bound is called a free variable. In case x does not appear freely in B, we abbreviate $\Pi x : A.B$ as $A \to B$. We furthermore use $A \equiv t$ to say A is a shorthand for term t.

In pCuIC, we consider terms equal up to renaming of bound variables, i.e., α-conversion. That is, a term $\lambda x : A.x$ is considered equal to $\lambda y : A.y$. We use $t[t'/x]$ to denote substitution of all *free* occurrences of variable x in term t with term t'. We, in addition, assume that, if necessary, before any substitution of free variables in a term α-conversion is performed so as to avoid any variable capture. In other words, before performing replacement $(\lambda y : A \to B.(y\ x))[y/x]$ we change it to $(\lambda z : A \to B.(z\ x))[y/x]$.

$$\textsc{Empty}\ \frac{}{\cdot \vdash} \qquad \textsc{Decl}\ \frac{\Gamma \vdash T : s \quad x \notin \Gamma}{\Gamma, x : T \vdash} \qquad \textsc{Type}\ \frac{\Gamma \vdash}{\Gamma \vdash \texttt{Type}_i : \texttt{Type}_{i+1}} \qquad \textsc{Prop}\ \frac{\Gamma \vdash}{\Gamma \vdash \texttt{Prop} : \texttt{Type}_i}$$

$$\textsc{Var}\ \frac{\Gamma \vdash \quad (x : T) \in \Gamma}{\Gamma \vdash x : T} \qquad \textsc{App}\ \frac{\Gamma \vdash t : (\Pi x : A.B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]}$$

$$\textsc{Prod}\ \frac{\Gamma \vdash A : s \quad \Gamma, x : A \vdash B : s' \quad (s, s', s'') \in R_\Pi}{\Gamma \vdash \Pi x : A.B : s''}$$

$$\textsc{Lam}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x : A.t) : (\Pi x : A.B)} \qquad \textsc{Conv}\ \frac{\Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \preceq B}{\Gamma \vdash t : B}$$

$$\textsc{Ind}\ \frac{A :: Ar(s) \quad (\Gamma, X : A \vdash C_i : s \quad C_i :: Co(X)\ \ \forall 1 \leq i \leq n)}{\Gamma \vdash \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\} : A}$$

$$\textsc{Constr}\ \frac{I \equiv \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\} \quad \Gamma \vdash I : A \quad 1 \leq i \leq n}{\Gamma \vdash \mathsf{Constr}(i, I) : C_i[I/X]}$$

$$\textsc{Elim}_{s,s'}\ \frac{\begin{array}{c} I \equiv \mathsf{Ind}(X : \Pi \vec{x} : \vec{A}.s)\{C_1, \ldots, C_n\} \quad \vec{a} : \vec{A} \quad (s, s') \in R_\xi \\ \Gamma \vdash c : (I\,\vec{a}) \quad \Gamma \vdash Q : \Pi \vec{x} : \vec{A}.(I\,\vec{x}) \to s' \quad (\Gamma \vdash f_i : \xi(I, Q, \mathsf{Constr}(i, I), C_i)\ \ \forall 1 \leq i \leq n) \end{array}}{\Gamma \vdash \mathsf{Elim}(c, Q)\{f_1, \ldots, f_n\} : (Q\ \vec{a})\ c}$$

Fig. 1. Typing Judgements

2.1 Typing Rules

The rules for typing judgements of this system are presented in Figure 1. There are two judgements in this figure, $\Gamma \vdash$ which denotes validity of a context Γ and $\Gamma \vdash t : A$ which indicates that term t has type A under context Γ. We shall call a term whose type is $\texttt{Prop}$ a *proposition* and a term whose type is a proposition a *proof*. Moreover, we shall call a term whose type is a sort a *type*. That is to say all propositions are types but not all types are propositions. This is to be understood under the Curry-Howard correspondence, a.k.a, *propositions as types*.

The relation R_{Π}, in Figure 1, governs the level of products formed in the system and is given by

$$R_{\Pi} = \{(\_, \texttt{Prop}, \texttt{Prop}), (\texttt{Type}_i, \texttt{Type}_j, \texttt{Type}_{max(i,j)})\}$$

In other words, $\texttt{Prop}$ is impredicative while $\texttt{Type}$ is predicative.

Rules Ind, Constr and Elim, respectively, pertain to formation of inductive types, their constructor terms and their eliminators. In these rules, arity for a sort s, $Ar(s)$, types strictly positive in X, $Pos(X)$ and types of constructors for X, $Co(X)$, are as follows:

$$\begin{aligned} Ar(s) &::= \Pi \vec{x} : \vec{M}.s \\ Pos(X) &::= \Pi \vec{x} : \vec{M}.X\,\vec{m} \\ Co(X) &::= X\,\vec{m} \mid Pos(X) \to Co(X) \mid \Pi x : M.Co(X) \end{aligned}$$

provided that in $Pos(X)$ and $Co(X)$, above, X does not appear in m or M. In this figure, relation R_{ξ} controls formation of eliminations and is defined as

$$R_{\xi} = \{(\texttt{Prop}, \texttt{Prop}), (\texttt{Type}_i, \texttt{Type}_j), (\texttt{Type}_i, \texttt{Prop})\}$$

That is, we do not allow terms that are not proofs to be constructed by case analysis on a proof.

In Figure 1, ξ is the type for eliminators defined below.

Definition 1 (Eliminator Type). *Let C be a type of constructor for X and let Q and c be two terms. Then, the type of eliminator for C, $\xi(I, Q, c, C) \equiv (\xi_X(Q, c, C))[I/X]$ is defined as follows:*

$$\begin{aligned} \xi_X(Q, c, P \to N) &= \Pi p : P.(\Pi \vec{x} : \vec{M}.(Q\ \vec{m}\ (p\ \vec{x}))) \to \xi_X(Q, (c\ p), N) \\ &\quad \textit{for } P \equiv \Pi \vec{x} : \vec{M}.(X\ \vec{m}) \\ \xi_X(Q, c, \Pi x : M.N) &= \Pi x : M.\xi_X(Q, (c\ x), N) \\ \xi_X(Q, c, X\ \vec{a}) &= (Q\ \vec{a}\ c) \end{aligned}$$

■

The following lemma states type correctness of eliminator types.

Lemma 1. *Let s_1, s_2 and s_3 be sorts, $A \equiv \Pi \vec{x} : \vec{A}.s_1$ be an arity for s_1 and C be a type of constructor for X such that $\Gamma, X : A \vdash C : s_1$ and $\Gamma, X : A \vdash (\Pi \vec{x} : \vec{A}.(X\ \vec{x}) \to s_2) : s_3$. Then, there is a sort s_4 such that:*

$$\Gamma, X : A, Q : \Pi \vec{x} : \vec{A}.(X\ \vec{x}) \to s_2, c : C \vdash \xi_X(Q, c, C) : s_4$$

■

Proof. By induction on the form of C. □

2.2 Reduction

The computational rule corresponding to inductive types, expectedly, corresponds to induction/recursion. The elimination of a term of an inductive type should perform a case analysis on its input and apply the corresponding provided elimination for that case by recursively eliminating any argument of the constructor that is of the inductive type. This will be made more clear later. For now let us consider recursors for constructors. A recursor for a constructor, as the name suggests, takes the arguments of a constructor and performs the provided elimination by recursively eliminating sub-terms. The recursor $\mu(I, F, f, C)$ for a constructor C of an inductive type I takes two terms f and F. The term f is the term that performs elimination for constructor C while term F corresponds to recursive elimination of sub-terms.

Definition 2 (Recursor). *Let C be a type of constructor for X and F and f be terms. Then, recursor $\mu(I, F, f, C) = (\mu_X(F, f, C))[I/X]$ is defined as follows:*

$$\begin{aligned}
\mu_X(F, f, P \to N) &= \lambda p : P.\mu_X(F, (f\ p\ (\lambda \vec{x} : \vec{M}.(F\ \vec{m}\ (p\ \vec{x})))), N) \\
&\quad \textit{for } P \equiv \Pi \vec{x} : \vec{M}.(X\ \vec{m}) \\
\mu_X(F, f, \Pi x : M.N) &= \lambda x : M.\mu_X(F, (f\ x), N) \\
\mu_X(F, f, X\ \vec{a}) &= f
\end{aligned}$$

■

The following lemma shows that recursors are well-typed when the terms provided to them for elimination of the corresponding constructor and recursive eliminations are.

Lemma 2. *Let s_1, s_2 and s_3 be sorts, $A \equiv \Pi \vec{x} : \vec{A}.s_1$ be an arity for s_1 and $C \equiv \Pi \vec{z} : \vec{B}.(X\ \vec{a})$ be a type of constructor for X such that $\Gamma, X : A \vdash C : s_1$ and $\Gamma \vdash (\Pi \vec{x} : \vec{A}.(X\ \vec{x}) \to s_2) : s_3$. Then,*

$$\begin{aligned}
&\Gamma, X : A, Q : \Pi \vec{x} : \vec{A}.(X \vec{x}) \to s_2, F : \Pi \vec{x} : \vec{A}.\Pi c : (X\ \vec{x}).(Q\ \vec{x}\ c), c : C, f : \xi_X(Q, c, C) \\
&\vdash \mu_X(F, f, C) : (\Pi \vec{z} : \vec{B}.(Q\ \vec{a}\ (c\ \vec{z})))
\end{aligned}$$

■

Proof. By induction on the form of C. □

We consider two computation rules for pCuIC, β, for function application,

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

and ι for elimination of inductive types,

$$\mathsf{Elim}((\mathsf{Constr}(i, I)\ \vec{m}), Q)\{f_1, \ldots, f_n\} \to_\iota (\mu(I, F_{elim}(I, Q, f_1, \ldots, f_n), f_i, C_i)\ \vec{m})$$

for $I \equiv \mathsf{Ind}(X : A)\{C_1, \ldots, C_n\}$, $A \equiv \Pi \vec{x} : \vec{A}.s$ where

$$F_{elim}(I, Q, f_1, \ldots, f_n) \equiv \lambda \vec{x} : \vec{A}.\lambda c : (I\ \vec{x}).\mathsf{Elim}(c, Q)\{f_1, \ldots, f_n\}$$

In rules β and ι above, we call the left hand side of the reductions, β-redex and ι-redex respectively. A term that has no redex is said to be in normal form. We say two terms M and N are $\beta\iota$-convertible, written as $M \simeq_{\beta\iota} N$ if there are terms $t_1, \ldots t_n$ such that we have

$$M \to^*_{\beta\iota} t_1\ {}^*_{\beta\iota}\!\leftarrow t_2 \to^*_{\beta\iota} \ldots\ {}^*_{\beta\iota}\!\leftarrow t_n \to^*_{\beta\iota} N$$

where $*$ denotes reflexive transitive closure. The convertibility relation is also referred to as *definitional equality*.

Another well-known rule for lambda calculus is η which corresponds, intuitively, to the principle of functional extensionality:

$$\lambda x : A.Mx \to_\eta M$$

However, it is known that adding η-reduction to a lambda calculus with sub-typing, like pCuIC, breaks the Church-Rosser property, i.e., uniqueness of normal forms. Simply consider the term $\lambda x : \mathtt{Type}_i.(\lambda y : \mathtt{Type}_{i+1}.y)\ x$. As we will see, considering the cumulativity relation discussed below, this term is well typed. Yet,

$$\lambda x : \mathtt{Type}_i.(\lambda y : \mathtt{Type}_{i+1}.y)\ x \to_\beta \lambda x : \mathtt{Type}_i.x$$

and

$$\lambda x : \mathtt{Type}_i.(\lambda y : \mathtt{Type}_{i+1}.y)\ x \to_\eta \lambda y : \mathtt{Type}_{i+1}.y$$

which are both in normal forms. Hence, this term would have two different normal forms. Nonetheless, we can consider η convertibility. We consider $\simeq_{\alpha\beta\iota\eta}$ as the conversion relation (definitional equality) for pCuIC. It is defined just like $\simeq_{\beta\iota}$ above with the addition of η-conversion and α-renaming. In the sequel, we shall use $\simeq$ and $\simeq_{\alpha\beta\iota\eta}$ interchangeably.

Before turning to the cumulativity relation, let us consider an example of inductive types and their elimination.

Example 1. Let us define in pCuIC the prime example of inductive types, natural numbers.

$$nat \equiv \mathsf{Ind}(X : \mathtt{Type}_0)\{X, X \to X\}$$

Let us use $Zero \equiv \mathsf{Constr}(1, nat) : nat$ and $Succ \equiv \mathsf{Constr}(2, nat) : nat \to nat$ to refer to the zero and successor constructors of the natural numbers. We construct the eliminator for type nat as follows, assuming $Q : nat \to s$.

$$\begin{array}{ll} \xi(nat, Q, X) = Q\ Zero & \text{for } Zero \\ \xi(nat, Q, X \to X) = \Pi p : nat.Q\ p \to Q\ (Succ\ p) & \text{for } Succ \end{array}$$

Consequently, we have:

$$Q : (nat \to s), f_1 : (Q\ Zero), f_2 : (\Pi p : nat.Q\ p \to Q\ (Succ\ p)), n : nat \vdash Elim(n, Q)\{f_1, f_2\} : Q\ n$$

By four applications of Lam, we can derive:

$$\begin{array}{l} \cdot \vdash \Big(\lambda Q : (nat \to s).\lambda f_1 : (Q\ Zero). \\ \quad \lambda f_2 : (\Pi p : nat.(Q\ p) \to Q\ (Succ\ p)).\lambda n : nat.Elim(n, Q)\{f_1, f_2\}\Big) : \\ \quad \Big(\Pi Q : (nat \to s).(Q\ Zero) \to (\Pi p : nat.(Q\ p) \to Q\ (Succ\ p)) \to \Pi n : nat.Q\ n\Big) \end{array}$$

Which is precisely the induction (in case $s = \mathtt{Prop}$) and recursion principle for natural numbers. As a concrete example, take addition for natural numbers:

$$add \equiv \lambda x : nat.\lambda y : nat.\mathsf{Elim}(x, \lambda z.nat)\{y, \lambda z.\lambda z'.(Succ\ z')\}$$

It is easy to see that

$$\vdash add : nat \to nat \to nat$$

is derivable. Moreover, we have:

$$\begin{array}{l} add\ Zero\ y \quad \to^*_{\beta\iota} y \\ add\ (Succ\ x)\ y \to^*_{\beta\iota} Succ\ \underbrace{(\mathsf{Elim}(x, \lambda z.nat)\{y, \lambda z.\lambda z'.(Succ\ z')\})}_{M} \end{array}$$

and

$$M \simeq add\ x\ y$$

■

For further reading on inductive types in the calculus of constructions refer to [3,4].

2.3 Cumulativity

The relation $\preceq$ in rule Conv reflects both convertibility and cumulativity. Rules for this relation are depicted in Figure 2. Rule C-Ind corresponds to cumulativity of inductive types. Intuitively,

$$\text{C-Prop}\ \frac{}{\mathtt{Prop} \preceq \mathtt{Type}_i} \qquad \text{C-Type}\ \frac{i \leq j}{\mathtt{Type}_i \preceq \mathtt{Type}_j} \qquad \text{C-Prod}\ \frac{A \simeq A' \quad B \preceq B'}{\Pi x : A.B \preceq \Pi x : A'.B'} \qquad \text{C-Conv}\ \frac{A \simeq B}{A \preceq B}$$

$$\text{C-Congr}\ \frac{A \simeq A' \quad A' \preceq B' \quad B \simeq B'}{A \preceq B}$$

$$\text{C-Ind}\ \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi \vec{x} : \vec{M_A}.s)\{\Pi \vec{x_1} : \vec{M_1}.X\ \vec{m_1}, \ldots, \Pi \vec{x_n} : \vec{M_n}.X\ \vec{m_n}\} \\ I' \equiv (\mathsf{Ind}(X : \Pi \vec{x} : \vec{M'_A}.s')\{\Pi \vec{x_1} : \vec{M'_1}.X\ \vec{m'_1}, \ldots, \Pi \vec{x_n} : \vec{M'_n}.X\ \vec{m'_n}\} \\ s \preceq s' \quad \forall i.\ (M_A)_i \preceq (M'_A)_i \quad \forall i,j.\ (M_i)_j \preceq (M'_i)_j \\ length(\vec{m}) = length(\vec{x}) \quad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

Fig. 2. Conversion/Cumulativity Relation

rule C-Ind establishes relation $I\ \vec{m} \preceq I'\ \vec{m}$, if every arity type and constructor parameter type of I is a subtype of the corresponding type in I'.

In pCuIC, Π types are considered invariant in their domain type. However, results similar to those discussed in this paper are generalizable to the case with full contravariance for the domain type of Π types.

Before discussing meta-theoretical properties of pCuIC, let us see some examples of cumulativity of inductive types.

Example 2. As an example, consider the type of categories which in pCuIC is of the form:

$$Category_{i,j} \equiv \mathsf{Ind}(X : \mathtt{Type}_{max(i+1,j+1)})\{\Pi o : \mathtt{Type}_i.\Pi h : o \to o \to \mathtt{Type}_j.N\}$$

for $i, j \in \mathbb{N}$ where i and j don't appear in N. Clearly, we can use C-Ind to derive $Category_{i,j} \preceq Category_{k,l}$ given that $i \leq k$ and $j \leq l$.

As another example, consider the type of ensembles. This type is expressed in pCuIC as follows:

$$Ens_i \equiv \mathsf{Ind}(X : \mathtt{Type}_{i+1})\{\Pi A : \mathtt{Type}_i.(A \to X) \to X\}$$

Again we can use C-Ind to derive $Ens_i \preceq Ens_j$ given that $i \leq j$. ■

As a result of cumulativity relations established in Example 2, none of the problems discussed earlier regarding the category of categories and the ensemble of ensembles occur in pCuIC.

3 Properties

Although we do not provide any proof, we believe that the following two propositions, stating properties of pCuIC and relation $\preceq$, respectively, hold and can be proven in a way akin to their counterparts in [2] or [3].

Proposition 1. *pCuIC has the following properties:*

1. *Church-Rosser Property for $\beta\iota$-reduction* (Church-Rosser)
2. *$\beta\iota$-strong normalization* (Strong Normalization)
3. *Every derivation of $\Gamma \vdash t : A$ has a sub-derivation that derives $\Gamma \vdash$ and every derivation of $\Gamma, x : T, \Gamma' \vdash$ has a sub-derivation that derives $\Gamma \vdash T : s$* (Context-Validity)
4. *if $\Gamma \vdash t : A$, then there is a sort s such that $\Gamma \vdash A : s$* (Typing-Validity)
5. *if $\Gamma \vdash t : A$, $\Gamma \subseteq \Delta$ and $\Delta \vdash$, then $\Delta \vdash t : A$.* (Weakening)
6. *if $\Gamma, x : T, \Gamma' \vdash P : A$ and $\Gamma \vdash M : T$, then $\Gamma, \Gamma'[M/x] \vdash P[M/x] : A[M/x]$* (Cut)
7. *if $\Gamma, x : A, \Gamma' \vdash M : C$, $\Gamma \vdash B : s$ and $B \preceq A$, then $\Gamma, x : B, \Gamma' \vdash M : C$* (Context Replacement)
8. *if $\Gamma, x : A, \Gamma' \vdash M : B$ and $x \notin FV(\Gamma') \cup FV(B) \cup FV(M)$, then $\Gamma, \Gamma' \vdash M : B$* (Strengthening)
9. *if $\Gamma \vdash t : A$ and $t \to^*_{\beta\iota} t'$ then $\Gamma \vdash t' : A$* (Subject Reduction)

■

Proposition 2. *Properties of conversion/cumulativity relation, $\preceq$:*

1. *$\preceq$ is a partial order relation up to $\simeq$:*

$$\frac{}{t \preceq t} \qquad \frac{t \preceq t' \quad t' \preceq t''}{t \preceq t''} \qquad \frac{t \preceq t' \quad t' \preceq t}{t \simeq t'}$$

2. *The relation $\preceq$ is well-founded, i.e., there is no infinite decreasing chain $A_0 \succ A_1 \succ \ldots$* (Well-Founded)
3. *if $\Gamma \vdash t : A$ then there exists B such that $\Gamma \vdash t : B$ and for any C such that $\Gamma \vdash t : C$ we have $B \preceq C$* (Principal Type)

■

The system presented in this paper, pCuIC, has a strictly richer type system compared to pCIC. In other words,

$$\Gamma \vdash_{\mathsf{pCIC}} t : A \text{ implies } \Gamma \vdash_{\mathsf{pCuIC}} t : A$$

but the converse does not hold. Consider the instance of the ensemble of small ensembles expressed in pCuIC as

$$\cdot \vdash (\mathsf{Constr}(1, Ens_{i+1})\ Ens_i\ (\lambda x : Ens_i.x)) : Ens_{i+1}$$

which is not directly typable in pCIC.

4 Lesser pCuIC

In this section we present the lesser pCuIC (lpCuIC). lpCuIC is a subsystem of pCuIC in which for any sub-derivation of the form $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$, we have $\Gamma \vdash_{\mathsf{pCIC}} T : s$ is derivable and for any sub-derivation of the form $\Gamma \vdash_{\mathsf{lpCuIC}} T : \Pi \vec{x} : \vec{A}.s$, we have $\Gamma \vdash_{\mathsf{pCIC}} T : \Pi \vec{x} : \vec{A}.s$ is derivable. In lpCuIC, rule C-Ind is replaced by:

$$\text{C-Ind'}\quad \frac{\begin{array}{c} I \equiv (\mathsf{Ind}(X : \Pi \vec{x} : \vec{M_A}.s)\{\Pi(\vec{x_1} : \vec{M_1}.X\ \vec{m_1}, \ldots, \Pi \vec{x_n} : \vec{M_n}.X\ \vec{m_n}\} \\ I' \equiv (\mathsf{Ind}(X : \Pi \vec{x} : \vec{M'_A}.s')\{\Pi \vec{x_1} : \vec{M'_1}.X\ \vec{m'_1}, \ldots, \Pi \vec{x_n} : \vec{M'_n}.X\ \vec{m'_n}\} \\ s \preceq s' \quad \forall i.\ (M_A)_i \preceq_{\mathsf{pCIC}} (M'_A)_i \quad \forall i,j.\ (M_i)_j \preceq_{\mathsf{pCIC}} (M'_i)_j \\ length(\vec{m}) = length(\vec{x}) \quad \forall i.\ X\ \vec{m_i} \simeq X\ \vec{m'_i} \end{array}}{I\ \vec{m} \preceq I'\ \vec{m}}$$

and rule App is replaced by:

$$\text{App'}\quad \frac{\Gamma \vdash t : (\Pi x : A.B) \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B[t'/x]} \quad (\Gamma \vdash_{\mathsf{pCIC}} t' : A \text{ or } x \notin FV(B))$$

That is to say, in establishing subtyping relation with C-Ind, we don't allow C-Ind to be used in any sub-derivations. Also, a function whose codomain type depends on the input may not be applied to a term that is not of the appropriate type in pCIC.

In lpCuIC, we define the following lifters for the cumulativity relation. These lifters are then used to show that any type inhabited in lpCuIC is also inhabited in pCIC. This will give us a soundness proof for lpCuIC. We furthermore discuss that lpCuIC, just like pCuIC, does not suffer from the problems mentioned in the introduction regarding the category of small categories and ensemble of small ensembles.

Definition 3 (Lifters). *Let T and T' be two terms such that $T \preceq_{\mathsf{lpCuIC}} T'$. Then, we define the lifter $\Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'}$ recursively on derivation of $T \preceq_{\mathsf{lpCuIC}} T'$. If the last rule used to derive $T \preceq_{\mathsf{lpCuIC}} T'$ is:*

C-Prop	*then*	$\Upsilon_{\mathtt{Prop} \preceq_{\mathsf{lpCuIC}} \mathtt{Type}_i} = \lambda x : \mathtt{Prop}.x$
C-Type	*then*	$\Upsilon_{\mathtt{Type}_i \preceq_{\mathsf{lpCuIC}} \mathtt{Type}_j} = \lambda x : \mathtt{Type}_i.x$
C-Prod	*then*	$\Upsilon_{\Pi x:A.B \preceq_{\mathsf{lpCuIC}} \Pi x:A'.B'} = \lambda f : \Pi x : A.B.\lambda x : A'.\Upsilon_{B \preceq_{\mathsf{lpCuIC}} B'}\ (f\ x)$
C-Conv	*then*	$\Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'} = \lambda x : T.x$
C-Congr	*then*	$\Upsilon_{A \preceq_{\mathsf{lpCuIC}} B} = \Upsilon_{A' \preceq_{\mathsf{lpCuIC}} B'}$
C-Ind	*then*	$\Upsilon_{I\ \vec{t} \preceq_{\mathsf{lpCuIC}} I'\ \vec{t}} = \lambda x : I\ \vec{t}.\mathsf{Elim}(x, Q)\{\phi_1, \ldots, \phi_n\}$
	for	$I \equiv \mathsf{Ind}(X : \Pi(\vec{x} : \vec{M_A}).s)\{C_1, \ldots, C_n\}$
		$I' \equiv \mathsf{Ind}(X : \Pi(\vec{x} : \vec{M'_A}).s')\{C'_1, \ldots, C'_n\}$
		$Q \equiv \lambda \vec{y} : \vec{M_A}.\lambda z : I\ \vec{y}.I'\ \vec{y}$
	and	$\phi_i = \upsilon(I, Q, \mathsf{Constr}(i, I), C_i, \mathsf{Constr}(i, I'), C'_i)$

Here, the constructor lifter for C, $\upsilon(I, Q, c, C, f, C') = \upsilon_X(Q, c, C, f, C')[I/X]$ is defined as follows:

$$\begin{aligned}
\upsilon_X(Q, c, P \to N, f, P' \to N') &= \lambda p : P.\lambda z : (\Pi \vec{x} : \vec{M}.Q\ \vec{t}\ (p\ \vec{x})).\\
&\quad \upsilon_X(Q, (c\ p), N, (f\ z), N')\\
&\quad \textit{for } P \equiv \Pi \vec{x} : \vec{M}.X\ \vec{t}\\
\upsilon_X(Q, c, \Pi x : M.N, f, \Pi x : M'.N') &= \lambda x : M.\upsilon_X(Q, (c\ x), N, (f\ x), N')\\
\upsilon_X(Q, c, X\ \vec{t}, f, X\ \vec{t}) &= f
\end{aligned}$$

■

Lemma 3 (Type Correctness of Constructor Lifters). *Let I and I' be two inductive types such that $I\ \vec{m} \preceq_{\mathsf{lpCuIC}} I'\ \vec{m}$. Let Γ be a context such that $\Gamma \vdash_{\mathsf{lpCuIC}} I\ \vec{m} : s$ and $\Gamma \vdash_{\mathsf{lpCuIC}} I'\ \vec{m} : s'$. Then, for any C and C', corresponding constructors of I and I' respectively, we have that*

$$\Gamma, X : \Pi \vec{x} : \vec{M_A}.s, c : C, f : C'[I'/X] \vdash_{\mathsf{pCIC}} \upsilon_X(Q, c, C, f, C') : \xi_X(Q, c, C)$$

for $I \equiv \mathsf{Ind}(X : \Pi(\vec{x} : \vec{M_A}).s)\{C_1, \ldots, C_n\}$ and $Q \equiv \lambda \vec{t} : \vec{M_A}.\lambda z : I\ \vec{t}.I'\ \vec{t}$ is derivable.

■

Proof. By induction on the form of C. □

Lemma 4 (Type Correctness of Lifters). *Let T and T' be two terms such that $T \preceq_{\mathsf{lpCuIC}} T'$ and $\Gamma \vdash_{\mathsf{lpCuIC}} T : s$ and $\Gamma \vdash_{\mathsf{lpCuIC}} T' : s'$. Then, $\Gamma \vdash_{\mathsf{pCIC}} \Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'} : T \to T'$.* ■

Proof. By induction on derivation $T \preceq_{\mathsf{lpCuIC}} T'$. The only interesting case is the case of C-Ind. That is, $T \equiv I\ \vec{m}$ and $T' \equiv I'\ \vec{m}$. In this case, let C and C' be corresponding constructors of I and I' respectively. It follows from Lemma 3 that:

$$\Gamma, c : C[I/X], f : C'[I'/X] \vdash_{\mathsf{pCIC}} \upsilon(I, c, C, f, C') : \xi(I, Q, c, C)$$

for $Q \equiv \lambda \vec{t} : \vec{M_A}.\lambda z : I\ \vec{t}.I'\ \vec{t}$. This allows us to derive:

$$\Gamma \vdash_{\mathsf{pCIC}} \phi_i : \xi(I, Q, \mathsf{Constr}(i, I), C_i)$$

and consequently:

$$\Gamma, x : T \vdash_{\mathsf{pCIC}} \mathsf{Elim}(x, Q)\{\phi_1, \ldots, \phi_n\} : T'$$

which allows us to finally derive:

$$\Gamma \vdash_{\mathsf{pCIC}} \lambda x : T.\mathsf{Elim}(x, Q)\{\phi_1, \ldots, \phi_n\} : T \to T'$$

which concludes the proof. □

Lemma 5 (Inhabitants in lpCuIC). *Let t and T be terms such that $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$. Then there exists t' such that $\Gamma \vdash_{\mathsf{pCIC}} t' : T$.* ■

Proof. By induction on derivation of $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$. The only interesting cases are App and Conv.

- App:
  In this case we have $\Gamma \vdash_{\mathsf{lpCuIC}} t : (\Pi x : A.B)$ and $\Gamma \vdash_{\mathsf{lpCuIC}} t' : A$. In this case there exists v such that $\Gamma \vdash_{\mathsf{pCIC}} v : (\Pi x : A.B)$. Note that $\Gamma \vdash_{\mathsf{pCIC}} B[t'/x] : s$, by definition of lpCuIC. In addition, by definition of lpCuIC, we have the following two cases:

- $\Gamma \vdash_{\mathsf{pCIC}} t' : A$:
  In this case we can derive:
  $$\Gamma \vdash_{\mathsf{pCIC}} (v\ t') : B[t'/x]$$
- $\Gamma \not\vdash_{\mathsf{pCIC}} t' : A$:
  In this case we have $x \notin FV(B)$ and by induction hypothesis there exists v' such that $\Gamma \vdash_{\mathsf{pCIC}} v' : A$. Thus, $B = B[t'/x] = B[v'/x]$ we can derive:
  $$\Gamma \vdash_{\mathsf{pCIC}} (v\ v') : B[v'/x]$$

– Conv:
In this case we have $\Gamma \vdash_{\mathsf{lpCuIC}} t : A$, $\Gamma \vdash_{\mathsf{lpCuIC}} B : s$ and $A \preceq_{\mathsf{lpCuIC}} B$. Note that in this case we have $\Gamma \vdash_{\mathsf{pCIC}} B : s$, due to definition of lpCuIC. By induction hypothesis we have $\Gamma \vdash_{\mathsf{pCIC}} v : A$. By Lemma 4, we have that $\Gamma \vdash_{\mathsf{pCIC}} \Upsilon_{A \preceq_{\mathsf{lpCuIC}} B} : A \to B$. Thus, we can derive $\Gamma \vdash_{\mathsf{pCIC}} \Upsilon_{A \preceq_{\mathsf{lpCuIC}} B}\ v : B$.

□

Corollary 1 (Soundness of lpCuIC).

$$\cdot \vdash_{\mathsf{lpCuIC}} t : \mathit{False} \text{ implies that there exists } t' \text{ such that } \cdot \vdash_{\mathsf{pCIC}} t' : \mathit{False}$$

where $\mathit{False} \equiv \mathsf{Ind}(X : \mathit{Prop})\{\}$. ■

Proof. Follows directly from Lemma 5. □

Lemma 6 (Computational Neutrality of Lifters). *Let T and T' be terms such that $T \preceq_{\mathsf{lpCuIC}} T'$ and $\Gamma \vdash_{\mathsf{pCIC}} \Upsilon_{T \preceq T'} : T \to T'$. Furthermore, let t be a term such that $\Gamma \vdash_{\mathsf{lpCuIC}} t : T$ and in this derivation, every sub-derivation of the form $\Gamma' \vdash c : A$ for $A \simeq \Pi \vec{x} : \vec{M_A}.I\ \vec{m}$ for some inductive type I, we have $c \simeq \lambda \vec{x} : \vec{M_A}.\mathsf{Constr}(i, I)\ \vec{v}$ for some i and $\vec{v}$. Then,*

$$\Upsilon_{T \preceq_{\mathsf{lpCuIC}} T'}\ t \to^*_{\beta\iota} t'$$

where t' differs from t only in some sorts. ■

Proof. By induction on derivation of $T \preceq_{\mathsf{lpCuIC}} T'$. The only interesting case is the case of C-Ind. In this case, we have $T \equiv I\ \vec{m}$ for some inductive type I. Thus, we must have $t \simeq \mathsf{Constr}(i, I)\ \vec{v}$ for some I and $\vec{v}$. Therefore, we can unfold $\Upsilon_{T \preceq T'}$ and apply ι. The result β-reduces to $\mathsf{Constr}(i, I')\ \vec{v'}$ where $\vec{v'}$ is $\vec{v}$ with $\Upsilon_{T \preceq T'}$ applied to some of its sub-terms. The result simply follows from the induction hypothesis. □

It is worth noting that the technique used for lpCuIC, i.e, liftings, is not generalizable to pCuIC. This is due to the fact that there are types in pCuIC that are not types in pCIC. As an example, consider the type $I \equiv \mathsf{Ind}(X : \Pi A : \mathtt{Type}_h.A \to \mathtt{Type}_h)\{X\}$. The term $T \equiv I\ \mathtt{Category}_{k,l}\ \mathsf{Constr}(i, \mathtt{Category}_{i,j})$, provided that the necessary conditions on i, j, k, l and h are satisfied, is a type in pCuIC but not in pCIC. We can easily see that

$$I\ \mathtt{Category}_{k,l}\ (\Upsilon_{\mathtt{Category}_{i,j} \preceq_{\mathsf{lpCuIC}} \mathtt{Category}_{k,l}}\ \mathsf{Constr}(i, \mathtt{Category}_{i,j}))$$

is a type in pCIC and we can construct a lifting from T to this type. Yet, in proving the counterpart of Lemma 5 for pCuIC, we will hit a problem in case App. Suppose we have an application $t\ t'$ where t is a function expecting a term of type T. In such a case, t would not

be applicable to the result of lifting t'. A similar situation can occur in case types in arity or arguments of constructors of an inductive type of this form.

Worthy of mention is also the fact that lpCuIC supports cumulativity for categories and ensembles and is therefore free of the problems mentioned regarding them at the beginning of this paper. Let us see an example of liftings discussed above.

Example 3. Consider, once more, the type of ensembles:

$$Ens_i \equiv \mathsf{Ind}(X : \mathtt{Type}_{i+1})\{\Pi A : \mathtt{Type}_i.(A \to X) \to X\}$$

Similarly to what we discussed in Example 2, we can derive $Ens_i \preceq_{\mathsf{lpCuIC}} Ens_{i+1}$. In this case,

$$\Upsilon_{Ens_i \preceq_{\mathsf{lpCuIC}} Ens_{i+1}} = \lambda x : Ens_i.\mathsf{Elim}(x, \lambda z : Ens_i.Ens_{i+1})\{\phi\}$$

for

$$\phi \equiv \lambda A : \mathtt{Type}_i.\lambda p : (A \to Ens).\lambda z : (A \to Ens_{i+1}).\mathsf{Constr}(1, Ens_{i+1})\ A\ z$$

Let $c \equiv \mathsf{Constr}(1, Ens_i)\ A\ f$ be a term such that $A : \mathtt{Type}_i$ and $f : A \to Ens_i$. Then,

$$\Upsilon_{Ens_i \preceq Ens_{i+1}} c \to^*_{\beta\iota} \mathsf{Constr}(1, Ens_{i+1})\ A\ (\lambda x : A.\Upsilon_{Ens_i \preceq Ens_{i+1}}\ (f\ x))$$

■

Example 3 shows that although the ensemble of small ensembles is not directly typable in pCIC, lifting it results in the ensemble of lifted small ensembles witch indeed is a term in pCIC.

5 Discussion and Conclusion

We presented pCuIC which extends pCIC with cumulativity for inductive types and discussed issues that this treatment helps mitigate. We, furthermore, justified the cumulativity relation for inductive types that we introduced by showing that there is a sub-system of pCuIC, lpCuIC, in which any such cumulativity relation has a corresponding lifting in pCIC. This, in addition, allowed us to reduce soundness of lpCuIC to the soundness of pCIC.

Inductive types considered lack parameters and mutual inductive types, see [4] for details. Parameters can be considered as variables in the context while an inductive type is being defined. For instance, consider the type of equality which is usually defined in pCIC as

$$eq \equiv \lambda A : \mathtt{Type}_i.\lambda x : A.\mathsf{Ind}(Z : A \to \mathtt{Prop})\{Z\ x\}$$

In general, the values of parameters can influence the variance of types involving them in an inductive definition. Consider the following:

$$F : \mathtt{Type}_i \to \mathtt{Type}_j \vdash \mathsf{Ind}(X : \mathtt{Type}_l)\{(F\ A) \to X\}$$

In this case, we can't determine, e.g., whether $F\ A \preceq F\ B$ for $A \preceq B$. Hence separate analysis of different instances of inductive types with different parameters can help make the cumulativity results more fine-grained. In a different approach, we could add support for variables in the context, e.g., F above, to specify variance of their result with respect to their input, if appropriate, in addition to their type.

On the other hand, mutually inductive types are restricted to only appear strictly positively in one another. Therefore, although it is subject to further research, it seems natural that the approach presented here can be straightforwardly extended to the case of mutual inductive types.

Another interesting case is when we have $x : \texttt{Type}_i$ in an inductive type. We have not considered variance of x in our relation. Doing so will result in having, e.g., $list\ A \preceq list\ B$ for $A \preceq B$. Such cumulativity relations can be very useful in practice, lessening the need of explicit conversions.

We believe that the typical ambiguity and also elaboration and unification algorithms presented in [5] can be directly extended to this system. However, as higher order unification is undecidable in general, lifting functions can be used as hints to facilitate unification when necessary. Note that these liftings are not based on case analysis on the input anymore and are hence free of the aforementioned problems.

References

1. Coquand, T., Paulin, C.: Inductively defined types. In: COLOG-88, International Conference on Computer Logic, Tallinn, USSR, Proceedings. pp. 50–66 (December 1988)
2. Luo, Z.: An Extended Calculus of Constructions. Ph.D. thesis, University of Edinbrugh, Department of Computer Science (June 1990)
3. Paulin-Mohring, C.: Inductive definitions in the system coq - rules and properties. In: International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, Proceedings. pp. 328–345 (March 1993)
4. Paulin-Mohring, C.: Introduction to the Calculus of Inductive Constructions (Nov 2014), `https://hal.inria.fr/hal-01094195`
5. Sozeau, M., Tabareau, N.: Universe polymorphism in coq. In: Interactive Theorem Proving, ITP 2014, Proceedings. pp. 499–514 (July 2014)